تایپ کستینگ

برخی مواقع در برنامه‌های خود ، نیاز داریم تا یک نوع داده‌ای را به نوعی دیگر تبدیل کنیم . مثلاً تبدیل یک عدد اینتجر به یک رشته . این کار را تایپ کستینگ Type Casting یا تبدیل نوع داده‌ای می نامند .

سه تابع درون ساخت برای پایتون وجود دارد که این کار را برای ما انجام می‌دهند . این توابع به شرح زیر می‌باشند :

```
int ()
float ()
str ()
```

تابع int در پایتون یک مقدار float یا یک مقدار رشته ای مناسب را گرفته و آن را به مقدار اینتجر تبدیل می‌کند . مثلاً برای تبدیل یک مقدار فلوت به اینتجر به صورت زیر عمل می‌کنیم :

```
int (5.712987)
```

که نتیجه آن می‌شود عدد ۵

برای تبدیل یک مقدار رشته ای به یک مقدار اینتجر می‌توانیم به صورت زیر عمل کنیم :

```
int ("4")
```

و نتیجه می‌شود عدد ۴ به صورت عدد (نه رشته ای)

مثال بالا رو دیدید ولی ما نمی‌توانیم بنویسیم int ("Hello") ویا int ("4.22321") چرا که بایستی مقدار رشته مناسب به ورودی بدهیم . در دو حالت خطا دریافت می‌کنیم .

تابع float در عوض یک مقدار اینتجر یا مقدار رشته ای مناسب را از ورودی گرفته و آن را به مقدار فلوت تبدیل می‌کند . برای نمونه اگر بنویسیم float (2) و یا float ("2") که نتیجه دریافت خروجی 2.0 می‌باشد . اگر که بنویسیم float ("2.09109") خروجی می‌شود 2.09109 که یک مقدار اعشاری ‹فلوت› می‌باشد و دیگر رشته ای نیست .

تابع str در دو سمت مقدار اینتجر یا فلوت را گرفته و خروجی رشته ای به ما می‌دهد .
برای مثال اگر بنویسیم str (2.1) خروجی می‌شود مقدار "2.1"
اکنون که شما سه نوع داده‌ای ابتدایی در پایتون را آموختید به انواع پیشرفته‌تر آن خواهیم پرداخت . در درس‌های بعدی با ما همراه شوید .

نوع داده‌ای لیست

لیست به مجموعه‌ای داده ای اشاره دارد که به صورت عادی به هم مرتبط اند . به جای ذخیره این داده‌ها به عنوان متغیرهای جداگانه ما می‌توانیم آن‌ها در یک لیست ذخیره کنیم . برای نمونه فرض کنید برنامه ما نیاز دارد که سن ۵ کاربر را ذخیره کند . به جای ذخیره سازی آن‌ها در پنج متغیر با نام های userAge1 تا userAge5 می‌توانیم آن‌ها را در قالب یک لیست ذخیره کنیم .

به منظور اعلان یک لیست از ساختار دستوری زیر استفاده می‌کنیم :

```
listName = [initial values]
```

به یاد داشته باشید که به منظور اعلان یک لیست در پایتون از براکت [] استفاده می‌کنیم . همچنین مقادیر لیست را با علامت کاما , از هم جدا می‌کنیم .

مثال :

```
userAge = [21, 22, 23, 24, 25]
```

ما همچنین می‌توانیم یک لیست را بدون تعیین هیچ مقدار اولیه‌ای اعلان کنیم . به این منظور به سادگی می نویسیم :

```
listName = []
```

اکنون یک لیست خالی بدون هیچ مقدار تعیین شده‌ای داریم . به منظور اضافه کردن مقادیر به لیست بایستی از تابع append استفاده کنیم که در زیر توضیح خواهیم داد . مقادیر تکی درون لیست توسط مقادیر ایندکس قابل دسترسی هستند . دقت کنید که ایندکس ها از مقدار صفر آغاز می‌شود نه از مقدار ۱ که این شیوه در اکثر زبان‌های برنامه نویسی رایج است مثل زبان C و یا Java

از آنجایی که اولین مقدار ایندکس صفر است مقدار دوم ۱ می‌باشد و الی آخر . مثلاً :

```
userAge [0] = 21
userAge [1] = 22
```

به صورت جایگزین شما می‌توانید مقادیر یک لیست را از آخر فراخوانی کنید . آخرین آیتم موجود در لیست ما ایندکس 1- و دومین آیتم از آخر ایندکس 2- را دارد . چگونه ؟ به صورت زیر:

```
userAge [-1] = 25
userAge [-2] = 24
```

شما می‌توانید یک لیست یا بخشی از آن را به یک متغیر اختصاص دهید . اگر که بنویسید

```
userAge2 = userAge
```

متغیر userAge2 می‌شود :

```
userAge = [21, 22, 23, 24, 25]
```

اگر که شما بنویسید :

```
userAge3 = userAge [2:4]
```

شما آیتم های دارای ایندکس ۲ را تا ایندکس ۱-۴ از لیست userAge برداشته و به userAge3 اختصاص می‌دهیم . یعنی از ایندکس ۲ تا ۳ که می‌شود :

```
userAge3 = [23, 24]
```

این نوع علامت گذاری 2:4 را slice یا برش می نامند . اگر از شیوه علامت گذاری برش در پایتون استفاده کنیم ، آیتم در ایندکس آغازین همیشه در لیست ما شامل می‌شود ولی آیتم آخرین همیشه از لیست حذف می‌شود . به همین دلیل است که نتیجه 2:4 به صورت زیر می‌شود :

```
userAge3 = [23, 24]
```

نه به صورت زیر :

```
user Age3 = [23, 24, 25]
```

نشانه گذاری slice دارای عدد سومی با نام Stepper می‌باشد . اگر که بنویسیم :

```
userAge4 = userAge[1:5:2]
```

در اینجا کاری که استپر انجام می‌دهد این است که هر زیر لیست شامل هر شماره دومی را از ایندکس ۱ تا ۴ به ما می‌دهد . در اینجا برای ما چونکه استپر عدد ۲ می‌باشد نتیجه می‌شود :

```
userAge4 = [22, 24]
```

علاوه براین نشانه گذاری اسلایس دارای مقادیر پیش‌فرض ‹Defaults› می باشد . مقدار پیش‌فرض برای اولین عدد صفر می‌باشد و مقدار پیش‌فرض برای عدد دوم اندازه برش داده شده لیست می‌باشد ! به مثال زیر توجه کنید واضح بیان شده است :

```
userAge [   :4 ]
```

در مثال بالا عدد اول آورده نشده و به جای آن مقدار پیش‌فرض یعنی صفر در نظر گرفته می‌شود یعنی اینکه از ایندکس صفر تا ایندکس ۱-۴ را به شما می‌دهد .

مثال دوم :

```
userAge[1:  ]
```

در مثال بالا عدم دوم آورده نشده است در نتیجه اندازه برش داده شده لیست در نظر گرفته می‌شود . اندازه اسلایس لیست ۱-۵ می‌باشد . (چرا که اندازه userAge۵ می‌باشد و دارای پنج آیتم می باشد) . پس مثال بالا از ایندکس ۱ تا ایندکس ۱-۵ یعنی ۱ تا ۴ را به ما می‌دهد .

برای ویرایش و تغییر ایتم ها در یک لیست می نویسیم :

```
listName [ ایندکس آیتم هایی که می‌خواهیم تغییر دهیم] = مقدار جدید
```

برای نمونه اگر بخواهیم که آیتم دوم را ویرایش کنیم به سادگی می نویسیم :

```
userAge [1] = 5
```

که در نتیجه عبارت بالا لیست شما به صورت زیر تغییر پیدا می‌کند :

```
userAge = [21, 5, 23, 24, 25]
```

برای اضافه کردن آیتم ها به لیست از تابع append استفاده می‌کنیم . برای مثال اگر بنویسیم :

```
userAge.append(99)
```

شما در حقیقت مقدار ۹۹ را به پایان لیست اضافه کرده‌اید . اکنون لیست شما به صورت زیر تغییر یافته است :

```
userAge = [21, 5, 23, 24, 25, 99]
```

برای حذف یک آیتم از لیست از ساختار دستوری زیر استفاده می‌کنیم :

```
del listName[ ایندکس آیتم هایی که می‌خواهیم از لیست حذف کنیم]
```

برای مثال اگر بنویسیم :

```
del userAge[2]
```

لیست به صورت زیر تغییر خواهد کرد (مقدار سومی با ایندکس ۲ حذف می شود)

```
userAge = [21, 5, 24, 25, 99]
```

برای اینکه در مبحث لیست ها خبره شوید برنامه زیر را اجرا کنید و به خروجی ها و نتایج بدست آمده دقت کنید :

```
#declaring the list, list elements can be of
different data types
myList = [1, 2, 3, 4, 5, "Hello"]
#print the entire list.
print(myList)
#You'll get [1, 2, 3, 4, 5, "Hello"]
#print the third item (recall: Index starts
from zero).
print(myList[2])

#You'll get 3
#print the last item.
print(myList[-1])
#You'll get "Hello"
```

```
#assign myList (from index 1 to 4) to
myList2 and print myList2
myList2 = myList[1:5]
print (myList2)
#You'll get [2, 3, 4, 5]
#modify the second item in myList and print
the updated list
myList[1] = 20
print(myList)
#You'll get [1, 20, 3, 4, 5, 'Hello']


#append a new item to myList and print the
updated list
myList.append("How are you")
print(myList)
#You'll get [1, 20, 3, 4, 5, 'Hello', 'How
are you']
#remove the sixth item from myList and print
the updated list
del myList[5]
print(myList)
#You'll get [1, 20, 3, 4, 5, 'How are you']
```

نوع داده‌ای تاپل

Tuples درست شبیه لیست ها می‌باشند با این تفاوت که شما نمی‌توانید مقادیر آن‌ها را ویرایش کنید . مقادیر اولیه که برای تاپل ها تعیین می کنید ، تا آخر برنامه ثابت باقی می‌مانند و قابل تغییر نیستند . کجا مفید هستند ؟ برای مثال برای ذخیره سازی اسامی ماه های سال درون برنامه مفید هستند .

برای اعلان یک تاپل ، از ساختار دستوری زیر استفاده کنید :

```
tupleName = (initial values)
```

دقت کنید که این بار به جای براکت از پرانتز برای اعلان تاپل استفاده کردیم . چندین مقدار را هم مثل قبل با کاما ، از هم جدا می کنیم . به عنوان مثال :

```
Example:
monthsOfYear = (“Jan”, “Feb”, “Mar”, “Apr”,
“May”, “Jun”, “Jul”, “Aug”,“Sep”, “Oct”,
“Nov”, “Dec”)
```

درست شبیه لیست به مقادیر تکی را از طریق ایندکس ها دسترسی پیدا می کنیم . مثلاً :

```
monthsOfYear[-1]    و    monthsOfYear[0]
```

مثال بالا به ترتیب از چپ مقادیر "Jan" و "Dec" را به ما می‌دهد .

نوع داده‌ای دیکشنری

دیکشنری مجموعه‌ای از جفت داده‌های به هم مرتبط می‌باشد . برای نمونه اگر که ما بخواهیم نام کاربری و سن ۵ کاربر را ذخیره سازی کنیم ، می‌توانیم از نوع داده‌ای دیکشنری استفاده کنیم.

به منظور اعلان یک دیکشنری از ساختار دستوری زیر استفاده می‌کنیم :

```
dictionaryName = {dictionary key : data}
```

در ساختار بالا نام دیکشنری را در سمت چپ تعیین کرده و در سمت راست کلید دیکشنری و داده مورد نظر را درون آکولاد تعریف می‌کنیم . توجه داشته باشید که کلید ها بایستی ‹درون یک دیکشنری› یگانه باشند . مثلاً شما نمی‌توانید یک دیکشنری به صورت زیر تعریف کنید :

```
myDictionary = {''Peter'' : 38, ''Jhon'' :
51, ''Peter'' : 13}
```

چرا ؟ به این دلیل که Peter دو بار به عنوان کلید درون یک دیکشنری استفاده شده است . دقت داشته باشید که برای اعلان دیکشنری از علامت آکولاد استفاده می‌کنیم .

همچنین چندین جفت را با استفاده از کاما از هم جدا می‌کنیم . برای مثال :

```
userNameAndAge = {''Emily'' : 28, ''Amanda''
: 26, Jack = 34, David = ''Not Available''}
```

همچنین شما می‌توانید یک دیکشنری را با استفاده از متد dict اعلان کنید . برای اعلان دیکشنری بالا با استفاده از متد dict به صورت زیر عمل می‌کنیم :

```
userNameAndAge = dict( Emily : 28, Amanda :
26, Jack = 34, David = ''Not Available''}
```

زمانی که از متد dict به منظور اعلان دیکشنری استفاده می‌کنیم ، به جای آکولاد از علامت پرانتز استفاده کرده و نیازی به قراردادن کلیدهای دیکشنری درون کوتیشن نمی باشد .

برای دسترسی یک آیتم درون دیکشنری از کلید دیکشنری استفاده می‌کنیم . برای مثال برای دسترسی به سن امیلی می نویسیم :

```
userNameAndAge [''Emily'']
```

که نتیجه آن می‌شود ۲۸ سال

برای ویرایش آیتم ها درون یک دیکشنری از ساختار دستوری زیر استفاده می‌کنیم :

```
dictionaryName [dictionary key of item to be
modified]
```

برای مثال به منظور ویرایش سن امیلی به صورت زیر عمل می‌کنیم :

```
userNameAndAge[''Emiy''] = 25
```

اکنون دیکشنری ما به صورت زیر تغییر می‌کند :

```
userNameAndAge = {''Emily'' : 25, ''Amanda''
: 26, Jack = 34, David = ''Not Available''}
```

همچنین ما می‌توانیم یک دیکشنری را بدون تعیین هیچ مقدار اولیه‌ای تعریف کنیم . به این منظور به سادگی به صورت زیر عمل کنیم :

```
dictionaryName = { }
```

در این حالت یک دیکشنری بدون هیچ آیتمی داریم .
به منظور اضافه کردن آیتم ها به یک دیکشنری از ساختار دستوری زیر استفاده می‌کنیم :

```
dictionaryName [dictionary key] = data
```

برای مثال اگر بخواهیم مشخصات ویکتوریا را به دیکشنری خود اضافه کنیم به صورت زیر عمل کنیم :

```
userNameAndAge [''Victoria''] = 45
```

در این وضعیت دیکشنری ما به صورت زیر تغییر می‌کند :

```
userNameAndAge = {''Emily'' : 25, ''Amanda''
: 26, Jack = 34, David = ''Not Available'',
''Victoria'' = 45}
```

به منظور حذف آیتم ها از یک دیکشنری از ساختار دستوری زیر استفاده می‌کنیم :

```
del dictionaryName [ dictionary key ]
```

برای نمونه به منظور حذف Amanda از دیکشنری به صورت زیر عمل می‌کنیم :

```
del userNameAndAge [''Amanda'']
```

اکنون دیکشنری ما به صورت زیر تغییر می‌کند و آماندا حذف می‌گردد :

```
userNameAndAge = {''Emily'' : 25, Jack = 34,
David = ''Not Available'', ''Victoria'' =
45}
```

به منظور تمرین بیشتر در این زمینه برنامه زیر را اجرا و خروجی را بررسی کنید :

```
#declaring the dictionary, dictionary keys
and data can be of different data
types
myDict = {“One”:1.35, 2.5:”Two Point Five”,
3:”+”, 7.9:2}
#print the entire dictionary
print(myDict)
#You’ll get {2.5: 'Two Point Five', 3: '+',
'One': 1.35, 7.9: 2}
#Note that items in a dictionary are not
stored in the same order as the way
you declare them.
#print the item with key = “One”.
print(myDict[“One”])
#You’ll get 1.35
#print the item with key = 7.9.
print(myDict[7.9])
#You’ll get 2
#modify the item with key = 2.5 and print
the updated dictionary
myDict[2.5] = “Two and a Half”

print(myDict)
#You’ll get {2.5: 'Two and a Half', 3: '+',
```

```
'One': 1.35, 7.9: 2}

#add a new item and print the updated
dictionary
myDict[“New item”] = “I’m new”
print(myDict)
#You’ll get {'New item': 'I’m new', 2.5:
'Two and a Half', 3: '+', 'One':
1.35, 7.9: 2}
#remove the item with key = “One” and print
the updated dictionary
del myDict[“One”]
print(myDict)
#You’ll get {'New item': 'I’m new', 2.5:
'Two and a Half', 3: '+', 7.9: 2}
```